Hypertree Decomposition for Parallel Constraint Solving

Ke Liu Sven Löffler Petra Hofstedt

liuke@b-tu.de
Brandenburg University of Technology Cottbus-Senftenberg

KPS 2017

Outline

Outline

Introduction

This presentation tries to briefly answer the following questions:

1. How do we solve a given constraint network in parallel?

Introduction

This presentation tries to briefly answer the following questions:

1. How do we solve a given constraint network in parallel?
   - Parallel search
   - Parallel consistency
   - **Parallel constraint solving**
2. Why do we need to do decomposition?

Introduction

This presentation tries to briefly answer the following questions:

1. How do we solve a given constraint network in parallel?
   - Parallel search
   - Parallel consistency
   - **Parallel constraint solving**
2. Why do we need to do decomposition?
   - If we divide the original intractable problem into the tractable sub-problems because many NP-complete and NP-hard problems can be solved in polynomial time if the corresponding hypergraph has the bounded hypertree-width [3].
   - The entire constraint network is dead-end free if it can achieve directional arc-consistency [1].

Introduction

This presentation tries to briefly answer the following questions:

1. How do we solve a given constraint network in parallel?
   - Parallel search
   - Parallel consistency
   - **Parallel constraint solving**
2. Why do we need to do decomposition?
   - If we divide the original intractable problem into the tractable sub-problems because many NP-complete and NP-hard problems can be solved in polynomial time if the corresponding hypergraph has the bounded hypertree-width [3].
   - The entire constraint network is dead-end free if it can achieve directional arc-consistency [1].

Introduction

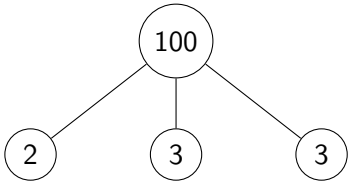
3 What is the motivation of the new decomposition algorithm?

- The target decomposition tree of existing algorithms are the hypertree with width as small as possible because the small hypertree width indicates the problem can be solved faster. The following situation could be happened!

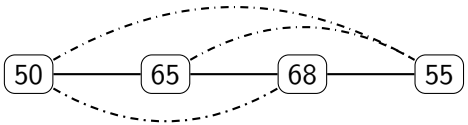
Introduction

3 What is the motivation of the new decomposition algorithm?

- The target decomposition tree of existing algorithms are the hypertree with width as small as possible because the small hypertree width indicates the problem can be solved faster. The following situation could be happened!



decomposed by det-k-decomp [3]



decomposed by det-k-CP

Introduction

4 How does Det-k-CP work?

- We will answer this question in the following slides.

Structural Decomposition Methods

The existing decomposition methods for constraint network

- Join-Tree-Clustering
- Cycle-cutset Decomposition
- Hinge Decomposition
- Hypertree Decomposition

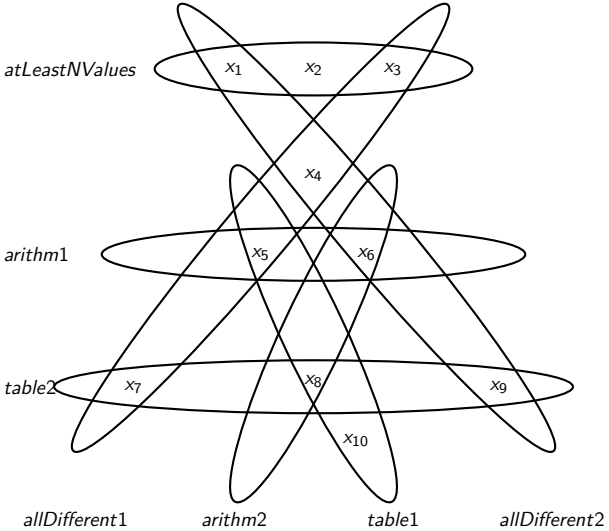
The main objective of Hypertree decomposition is to obtain a decomposition graph as small hypertree width as possible.

Hypertree decomposition example

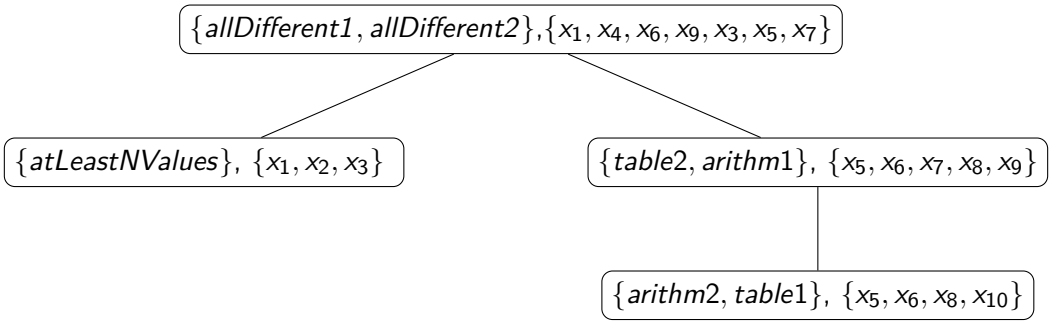
- $allDifferent1(x_3, x_4, x_5, x_7)$
- $allDifferent2(x_1, x_4, x_6, x_9)$
- $atLeastNvalues(x_1, x_2, x_3)$
- $arithm2(x_6, x_8)$
- $table1(x_5, x_8, x_{10})$
- $table2(x_7, x_8, x_9)$
- $arithm1(x_5, x_6)$

The hypergraph for the constraint network.

The example is based on [2]



Hypertree decomposition



One possible hypertree decomposition for the hypergraph on the previous page

Outline

The Objective of det-k-CP

- The design objective of *det-k-CP* is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time

The Objective of det-k-CP

- The design objective of *det-k-CP* is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time
  - Relatively even distribution for multi-core processor.

The Objective of det-k-CP

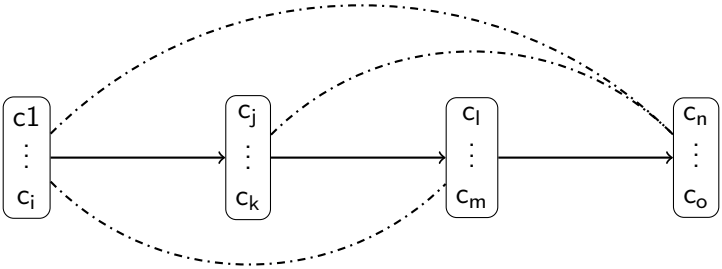
- The design objective of *det-k-CP* is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time
  - Relatively even distribution for multi-core processor.
  - **Accords with hypertree structure**

The Objective of det-k-CP

- The design objective of *det-k-CP* is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time
  - Relatively even distribution for multi-core processor.
  - **Accords with hypertree structure**

The Objective of det-k-CP

- The design objective of *det-k-CP* is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time
  - Relatively even distribution for multi-core processor.
  - **Accords with hypertree structure**



A degenerate decomposition tree for parallel solving.

A target degenerate decomposition tree

An acyclic constraint network can be solved efficiently. [1]

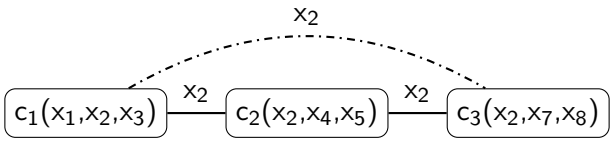
- *Because the constraint problem has a join-tree, its dual problem is a tree of binary constraints and can therefore be solved by the tree-solving algorithm* [1].

An Important Property

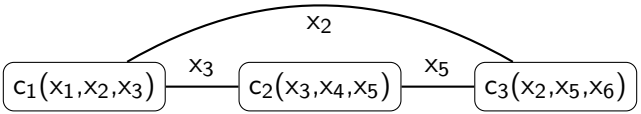
One edge between two nodes can be eliminated with-out changing the set of all solutions for the constraint network because there exists an alternative path between the two nodes.

An Important Property

One edge between two nodes can be eliminated with-out changing the set of all solutions for the constraint network because there exists an alternative path between the two nodes.

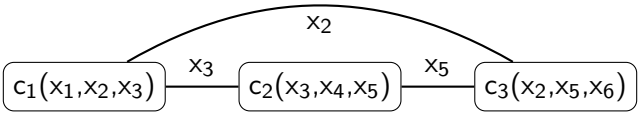


The Principle of det-k-CP

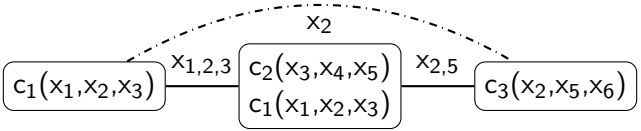


The original simple constraint network

The Principle of det-k-CP

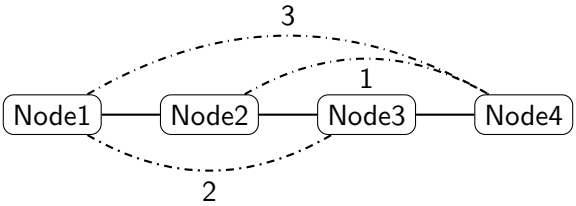


The original simple constraint network



Adding procedure for eliminating non-adjacent edge

The problem caused by simply adding constraint



The arc that had been removed before might occur again due to the subsequent adding procedure.

Outline

Two conditions of Det-k-CP

Mathematically, a decomposed graph after decomposition by *det-k-CP* must meet the following two conditions:

- The shared variables between any pair of nodes (N_p, N_{p+1}) in the interval $[i..j]$ must contain the shared variables between (N_i, N_j), where $i \leq p < p+1 \leq j$.

Two conditions of Det-k-CP

Mathematically, a decomposed graph after decomposition by *det-k-CP* must meet the following two conditions:

- The shared variables between any pair of nodes (N_p, N_{p+1}) in the interval $[i..j]$ must contain the shared variables between (N_i, N_j), where $i \leq p < p+1 \leq j$.
- The decomposition method *det-k-CP* does not lose any constraint or variable.

The main procedure of Det-k-CP

Algorithm 1: The main procedure of Det-k-CP

Input: A Constraint Network N, and the desired number of nodes k.

```
1  Set a list which contains sorted constraints of N based on weight.
2  Evenly distribute constraint onto k nodes.
3  while true do
4  |   get a Potential Solution;
5  |   if The graph passes test conditions (1) and (2) then
6  |   |   break;
7  |   end
8  |   Swap nodes in array_Nodes;
9  end
10 return a degenerated decomposition tree;
```

Get a potential solution for Det-k-CP

Algorithm 2: getPotentialSolution

Output: A potential solution

```
1 Set i_len = the number of nodes;
2 Set i_start = i_len-2;
3 while i_start ≥ 0 do
4     for i_end ← i_start + 2 to i_len do
5         eliminateEdge(i_start, i_end, array_Nodes) ;
6     end
7     Set i_start = i_start − 1 ;
8 end
```

Eliminate an Edge in Det-k-CP

Algorithm 3: eliminate an Edge between node

Input: *i_start*, *i_end*, *array_Nodes*

```
1  for i ← i_start to i_end − 1 do
2      Set list_2BeEliminated = getSharedVariables(i_start, i_end, array_Nodes);
3      Set j = i + 1 ;
4      Set list_SharedOnMainPath = getSharedVariables(i, j, array_Nodes);
5      if list_SharedOnMainPath.notContainsAll(list_2beEliminate) then
6          list_2BeEliminated.removeAll(list_SharedOnMainPath);
7          Set list_2beAddedConstraints = getMinimumSetConstraints4Shared(i, array_Nodes,
             list_2BeEliminated);
8          foreach constraint cs ∈ list_2beAddedConstraints do
9              if array_Nodes[j] notContained cs then
10                 add cs into array_Nodes[j];
11             end
12         end
13     end
14 end
```

Time complexity of Det-k-CP

The asymptotic time of Det-k-CP is $\mathcal{O}(k^4 \cdot N_c \cdot (N_c + N_v))$, where k is the number of nodes in the desired degenerated decomposition tree, N_c and N_v are the number of constraints and the number of variables for the given constraint network respectively.

Outline

Experiment setup

- The benchmark suit for testing det-k-CP is extracted from practical industrial constraint problem [3].
- The algorithms are implemented in Java.
- All the experiments are set up on an iMac computer having an Intel i7-3770 CPU, 3.40GHz, with 8 GB 1600 MHz DDR3.

Results and Conclusion of the experiment

- For the application of parallel constraint solving, the algorithm can be applied to medium scale constraints network with the number of constraints ranging from around 400 to 700.

Results and Conclusion of the experiment

- For the application of parallel constraint solving, the algorithm can be applied to medium scale constraints network with the number of constraints ranging from around 400 to 700.
- Competitive execution time as long as the hypergraphs are sufficiently simple. e.g., the constraint network *s5378* with 2958 constraints and 2993 variables) and a big k (e.g., k = 16) takes around two minutes.

Conclusion and Future Work

- The possible ways for improving det-k-CP
  - Take into consideration an estimate of the amount/complexity of computation for the constraints
  - Local search methods might be employed to replace the existing stochastic strategies in order to obtain more optimized decomposition result.
  - Add constraints to one node from other nodes covering the same variables.

Conclusion and Future Work

- The possible ways for improving det-k-CP
  - Take into consideration an estimate of the amount/complexity of computation for the constraints
  - Local search methods might be employed to replace the existing stochastic strategies in order to obtain more optimized decomposition result.
  - Add constraints to one node from other nodes covering the same variables.
- Future Work
  - To develop a related join selection algorithm for det-k-CP to gain the speed up.

Thank you for your attention!
Question?

References

Rina Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

Georg Gottlob, Nicola Leone, and Francesco Scarcello. "Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width". In: *Journal of Computer and System Sciences* 66.4 (2003), pp. 775–808.

Georg Gottlob and Marko Samer. "A backtracking-based algorithm for hypertree decomposition". In: *Journal of Experimental Algorithmics (JEA)* 13 (2009), p. 1.